

Foliage Discrimination using a Rotating Ladar

Andres Castano Larry Matthies
 Jet Propulsion Laboratory
 4800 Oak Grove Dr. MS 125-209
 Pasadena, CA 91109

Abstract

An outdoor environment presents to a robot objects that are drivable, such as tall grass and small bushes, or non-drivable, such as trees and rocks. Due to the difficulty of discriminating between these classes, traditionally a robot searches for paths free of any objects drivable or not. Although this approach prevents collisions with objects misclassified as drivable, it also eliminates a large number of drivable paths and by doing so, it may eliminate the only path to a desired destination. We present a real time algorithm that detects foliage using range from a rotating laser. Objects not classified as foliage are conservatively labeled as non-drivable obstacles. In contrast to related work that uses range statistics to classify the objects, we exploit the expected localities and continuities of an obstacle, in both space and time. Also, instead of attempting to find a single accurate discriminating factor for every ladar return, we hypothesize the class of some few returns and then spread the confidence (and classification) to other returns using the locality constraints. The Urbie robot is presently using this algorithm to discriminate drivable grass from obstacles during outdoor autonomous navigation tasks.

I. Introduction

A major problem for robots during autonomous cross-country navigation is the discrimination between drivable and non-drivable objects in the path. Until a few years ago, outdoor robotics was mostly avoided, in part because many core research problems (e.g., path planning, mapping, etc.) could be studied indoors, decoupling them from the uncertainty associated with unstructured environments. The main exception was outdoor navigation of man-made roads, as in the case of the Navlab vehicle and its successors [1]. As the state of all-terrain sensors improves, the development of cross-country navigation platforms, like military surveillance robots and search and rescue vehicles, is becoming cost-effective.

The laser radar (ladar) is one of the sensors that is becoming cost-effective in cross-country navigation. The ladar is an active sensor that fires a laser beam and then senses its reflection, or return, from the scene. From this return, the sensor estimates the distance, or range, to the object in the scene hit by the beam. The use of a rotating mirror allows the sensor to sweep the scene about an axis, obtaining a 1-D range signal (e.g., [2]); placing a 1-D laser on a tilt unit allows the sensor to sweep an area of the scene, producing a range image (e.g., [3], [4]). Ladars are useful because they provide a range estimate in many situations where it cannot be estimated with a stereo pair

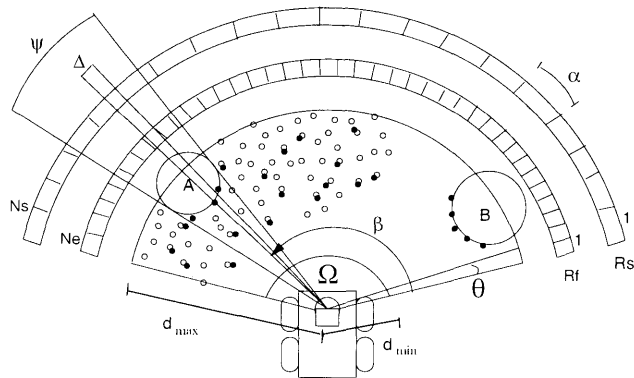


Fig. 1. Robot using a ladar to scan a scene

or by other means, e.g., night operation, low-frequency high-contrast scenes with shadows, etc.

In this paper we describe a real time algorithm that uses the range estimates of a rotating ladar to detect foliage in an outdoors scenario. This classification allows the inclusion of tall grass to any free path that the robot may select while still allowing it to avoid partially hidden obstacles. Recent approaches to using a ladar to find obstacles partially hidden by grass rely on the statistics of the signal (e.g., [5], [6]). In contrast to these approaches, we exploit the spatial and temporal localities of the objects in the scene to classify the returns. The result is a robust algorithm with a low false alarm rate.

This paper is organized as follows. In Secs. II and III we describe the ladar and provide an overview of the algorithm. In Secs. IV and V we show how to select non-foliage returns and how to prune them, respectively. In Sec. VI we describe how we keep track of the state of the scene. In Sec. VII we describe experimental results using Urbie, an all-terrain autonomous robot. Finally, in Sec. VIII we present our conclusions.

II. The Ladar System

Consider a mobile robot moving in a field of grass, as shown in Fig. 1. This figure shows a top view of a cross-section of the scene, where blades of grass and tree trunks are represented by small and large white circles, respectively. Tree trunks A and B represent the cases where an obstacle is partially hidden by grass and in the clear, respectively. The robot scans the scene parallel to the ground, once every τ seconds. Each scan spans an angle of Ω radians and provides range estimates to objects located farther than d_{min} and closer than d_{max} . Thus, a sampling interval (i.e., the angle between successive fires of the laser) of θ radians yields at most

$$N_e = \Omega/\theta$$

range estimates per scan. Some estimates might be missing, e.g., there might not be an object in the path of the laser within the valid distances or the target might absorb the light at the wavelength of the laser. In Fig. 1, the locations of the scene hit by a beam of the scan are marked with a black circle.

The accuracy of each range estimate is affected by parameters as varied as distance, environmental temperature and color and pose of the target [2]. In practice, within a narrow range interval, the error δ_{d_i} of an estimate can be considered to vary linearly with the distance d_i . Thus, the range estimate of an object located at a distance d_i is

$$r_i = d_i \pm (a d_i + b)$$

with

$$a = \frac{\delta_{max} - \delta_{min}}{d_{max} - d_{min}} \quad \text{and} \quad b = \delta_{min} - a d_{min}, \quad (1)$$

for $d_{min} \leq d_i \leq d_{max}$.

Finally, we consider negligible the laser divergence within the valid range. Thus, the beam is so narrow that it cannot hit two different objects located at different distances which creates erroneous range estimates formed by a combination of both distances.

III. Algorithm Overview

The goal of the algorithm is to discriminate foliage from other elements of the scene by classifying every return of each scan as FOLIAGE or NOTFOLIAGE. After each scan is processed, the algorithm produces three outputs that describe the results at different levels of detail. The most detailed output is the array of final range estimates, R_f , which contains N_e elements, each one corresponding to one return of the scan, as shown in Fig. 1. If we are not interested in the classification of each return then the array R_s summarizes the results that span a given angle α . Hence, R_s has N_s elements where

$$N_s = \Omega/\alpha.$$

Finally, the most general output is the flag *alarm* which is set to indicate that there is an object in the scene that is not foliage.

The pseudo-code of the algorithm is:

```

ANALYZE-LADAR-SCANS ()
1   $R_f(1 : N_e) \leftarrow 0$ 
2   $R_s(1 : N_s) \leftarrow 0$ 
3  forever
4     $[A, R] \leftarrow \text{GET-SCAN-DATA}()$ 
5     $[alarm, R_s, R_f] \leftarrow \text{CLASSIFY}(A, R)$ 

```

After initializing the arrays R_f and R_s we proceed to process the scans. Line 4 reads the scan into two arrays of N_e elements, A and R . The element $A(i)$ is the angle at which the i -th beam was fired while $R(i)$ is the raw range estimate of the object hit by the beam. In the rest of this paper we assume that we have access to N_e , N_s , d_{min} and d_{max} .

In the routine CLASSIFY(), we exploit three locality principles to find NOTFOLIAGE returns. To illustrate

these principles we assume that at time t we found an obstacle at angle β , e.g., the tree trunk A in Fig. 1. First, the locality in time of the obstacle indicates that it will be located around β at time $t+\tau$. Second, we use the locality in space of the obstacle, i.e., an obstacle must have a large size, spanning over a large angle ψ . Thus, if a beam hits the obstacle at angle β then all beams that might hit the obstacle must lie within $\beta \pm \psi$. Finally, we use the locality in space of the clear in the foliage that allowed the laser to hit the partially hidden obstacle, i.e., if a beam at angle β penetrates the foliage, then all its immediate neighbors fired at $\beta \pm \Delta$, for $\Delta \ll \psi$, are likely to penetrate the foliage through this same clear too. These locality principles hold for any combination of motions of the robot and the obstacles, as long as the sampling interval, θ , and the time between consecutive scans, τ , are sufficiently small.

The pseudo-code of the CLASSIFY routine is

```

CLASSIFY(A, R)
1   $[v, nad2v] \leftarrow \text{FIND-LOW-FREQ}(A, R)$ 
2   $R_c \leftarrow \text{GET-OBST-CANDIDATES}(A, R, v, nad2v)$ 
3   $R_c \leftarrow \text{RELAX-OBST}(A, R, R_c)$ 
4   $R_f \leftarrow \text{REMOVE-NEW-OBST}(A, R_c)$ 
5   $R_f \leftarrow \text{REMOVE-THIN-OBST}(R_f, W_s, O_s)$ 
6   $R_f \leftarrow \text{CROP-RANGE}(R_f, d_{min}, d_{max})$ 
7   $[alarm, R_s] \leftarrow \text{UPDATE-AND-ARCHIVE}(A, R_c)$ 
8  return ( $alarm, R_s, R_f$ )

```

The first three routines determine R_c , an array of returns likely to have hit obstacles. The second three routines determine R_f , that classifies each return of the scan. Finally, the routine UPDATE-AND-ARCHIVE saves the results for evaluation of future scans. We now discuss these routines.

IV. Candidate Selection

The first three routines of CLASSIFY select returns that are likely to belong to obstacles.

A. Finding low-frequency scan ranges

The routine FIND-LOW-FREQ uses the estimates R to find an array R_c where it is easy to identify returns that belong to obstacles. Consider the returns in Fig. 1. If we plot R , as shown in Fig. 2.a, we notice that the problem of locating the obstacles amid the foliage is similar to that of recovering a signal buried in noise. Thus, the obstacles can be recovered better if we filter out some of the foliage returns, increasing the signal-to-noise ratio of the scan.

To select a filter that removes mostly foliage returns, consider all the possible returns within an angle α (i.e., the size of our filter) that are centered around an angle β , as shown in Fig. 1. If the obstacle spans an angle larger than α , any beam fired at $\beta \pm \alpha/2$ can travel, at most, as far as the obstacle. Thus, the largest return of the set is likely to belong to the obstacle if any of the beams hit the obstacle. Hence, in our case, we can use a maximum-value filter that will remove mainly foliage returns. The size of the filter, α , must be larger than the sampling interval θ and smaller than the angle spanned by the smallest object that we want to detect when located at a distance d_{min} from the sensor. The

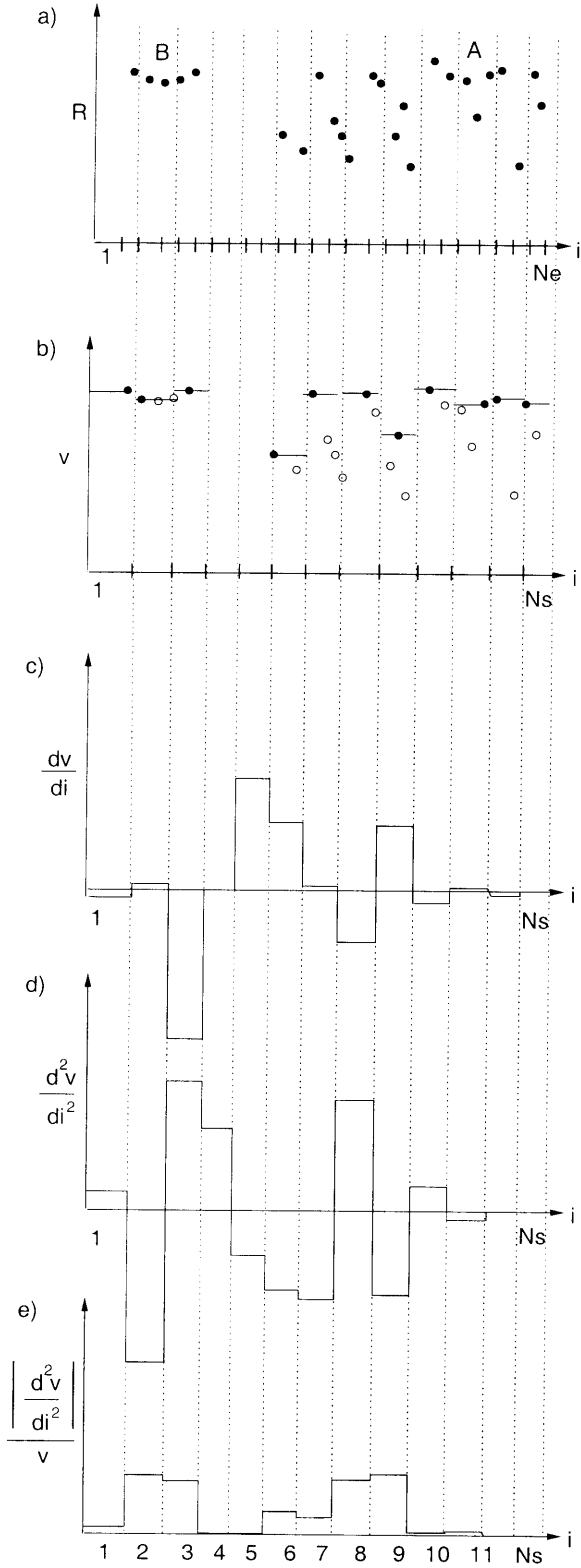


Fig. 2. a) The estimates of the example shown in Fig. 1, b) the array v of maximum range values, c) its derivative, d) second derivative and e) the absolute value of its second derivative divided by v

result of applying such filter to nonoverlapping windows of R is shown in Fig. 2.b. Within each window, a black dot indicates the maximum return selected by the filter, a white dot indicates a return filtered out, and a solid line indicates the value of the resulting filtered scan v . Since, in this example, $\alpha \approx 2\theta$ then v has $N_s \approx N_e/2$ elements.

The next step is to identify low frequency regions of the array v that might indicate the presence of an obstacle using the magnitude of its first or second derivatives. In Figs. 2.c-d we show approximations of these derivatives for the example, found using forward differences. If the face of the obstacle, as mapped on the array v , is either constant or follows a ramp function, then the magnitude of the second derivative is small. Thus, we are able to identify objects of any shape and in any pose with respect to the sensor, as long as their surfaces are locally flat, by searching for areas of small magnitude in the second derivative.

The last step of this routine is to generate the array $nad2v$ which stands for normalized absolute value of the second derivative of v . The absolute value is used to allow a magnitude-based thresholding of the second derivative of v . Furthermore, a normalization of $nad2v$ by the distance to the object, removes the bias introduced by the fact that objects located at different distances have a different associated noise that affects the measure of their frequency.

The pseudo-code of the FIND-LOW-FREQ routine is

```

FIND-LOW-FREQ ( $A, R$ )
1   $v(1 : N_s) \leftarrow 0$ 
2  for  $r \leftarrow 1$  to  $N_e$ 
3     $i \leftarrow \lfloor A(r)/\alpha \rfloor + 1$ 
4    if  $R(r) > v(i)$ 
5       $v(i) \leftarrow R(r)$ 
6  for  $i \leftarrow 1$  to  $N_s - 1$ 
7     $dv \leftarrow v(i+1) - v(i)$ 
8  for  $i \leftarrow 1$  to  $N_s - 2$ 
9    if  $v(i) = 0$ 
10      $nad2v \leftarrow 0$ 
11   else
12      $nad2v \leftarrow |dv(i+1) - dv(i)| / v(i)$ 
13  return ( $v, nad2v$ )

```

Lines 1-5 filter R and generate v . Lines 6-7 approximate the derivative of v using forward differences and, finally, lines 8-12 generate the absolute value of the second derivative of v normalized by v .

B. Selecting candidates

The GET-OBST-CANDIDATES routine selects from R those returns that are most likely to be obstacles returns. The pseudo-code of this routine is

```

GET-OBST-CANDIDATES ( $A, R, v, nad2v$ )
1   $R_c(1 : N_e) \leftarrow 0$ 
2  for  $i \leftarrow 1$  to  $N_s$ 
3     $T_k(i) \leftarrow K_1 + \text{MAX}(T_{k-1}(i), T_{k-2}(i))/2$ 
4   $done \leftarrow 0$ 
5  while  $done = 0$ 
6     $done \leftarrow 1$ 
7    for  $r \leftarrow 3$  to  $N_e - 2$ 
8       $i \leftarrow \lfloor A(r)/\alpha \rfloor + 1$ 

```

```

9   if  $v(i) = 0$  or  $R(r) = R_c(r)$  or  $R(r) = 0$ 
10  continue
11  if  $(R(r) = v(i))$  and
     $nad2v(i) = \min(nad2v(i), T_k(j)|_{j=i-2}^{i+2})$ 
12     $R_c(r) \leftarrow R(r)$ 
13     $T_k(i) = T_k(i) + \delta$ 
14     $T_k(i \pm 1) = T_k(i \pm 1) + \delta/2$ 
15     $T_k(i \pm 2) = T_k(i \pm 2) + \delta/3$ 
16     $done \leftarrow 0$ 
17   $T_{k-2} \leftarrow T_{k-1}$ 
18   $T_{k-1} \leftarrow T_k$ 
19  return  $(R_c)$ 

```

It returns an array of candidates R_c which is zero except at the locations of returns for which there is a strong evidence that belong to obstacles.

A small magnitude of $nad2v(i)$ indicates a signal with low frequency components which might indicate the presence of obstacles. For example, in the plot of $nad2v$ shown in Fig. 2.e, of the five locations with a small magnitude, the locations 1 and 10 correspond to obstacles while 4 and 5 correspond to low frequency areas caused by a lack of signal. Thus, an obstacle in an area i can be observed if the value of $nad2v(i)$ is smaller than a threshold $T_k(i)$. In lines 2-3 we set this threshold to the sum of a constant K_1 and a function of the thresholds found for this location in the two previous scans. We describe the function that we used in Line 3; other functions that raise the threshold at locations where thresholds were large in previous scans could also be suitable.

The main part of the routine updates the arrays of candidates R_c and thresholds T_k . A fast-rejection condition, in line 9, rejects those areas of the scan with no returns (i.e., $v(i) = 0$), individual locations with no return (i.e., $R(r) = 0$) and locations that already contain a candidate (i.e., $R(r) = R_c(r)$). The update condition, in line 11, states that a return is likely to be from an obstacle if it was selected by the maximum-value filter (i.e., $R(r) = v(i)$) and, if together with its four closest neighbors, it spans a low frequency region (i.e., $nad2v$ is smaller than all the thresholds within a vicinity of 2). If the update condition is verified, the array of candidates is updated with the value of the estimate (i.e., $R_c(r) = R(r)$) and, having detected this position as an obstacle, all the thresholds of the neighborhood are raised by some fraction of a value δ . As shown in lines 13-15, we raise more the thresholds of locations close to the identified obstacle than those away from it.

C. Reevaluating returns near to candidates

The non-zero locations of R_c are very likely to be obstacles but to achieve this confidence we sacrificed resolution that we now must recover. At this point, a value of R_c can only be a candidate if it has the value selected by the maximum-value filter. The goal of RELAX-OBST is to use the location of each obstacle in R_c to reevaluate the returns immediately adjacent to it. The pseudo-code of this routine is

RELAX-OBST(A, R, R_c)

```

1   done  $\leftarrow 0$ 
2   while done = 0
3     done  $\leftarrow 1$ 
4     for  $r \leftarrow 3$  to  $N_e - 2$ 
5        $i \leftarrow \lfloor A(r)/\alpha \rfloor$ 
6        $C \leftarrow 1 + K_2(C_{k-1}(i) + C_{k-2}(i))$ 
7        $T \leftarrow C(aR(r) + b)$ 
8       if  $R_c(r) = 0$  and
         $\bigcup_{j=-2}^2 ((R_c(r+j) > 0 \text{ and } |R(r) - R(r+j)| < T))$ 
9          $R_c(r) \leftarrow R(r)$ 
10      done  $\leftarrow 0$ 
11  return  $(R_c)$ 

```

The main loop updates the array of candidates R_c until there is a reevaluation of the returns of R that does not lead to an update of R_c . As indicated in lines 8-9, a return becomes a candidate if it was not previously a candidate and if any of its adjacent returns is both already a candidate and is such that the difference between the two returns is smaller than a threshold T . Hence, we use the locality in space of the obstacles to reevaluate a return adjacent to a return that is known to belong to an obstacle; if two adjacent beam hit targets located at about the same distance and if we know that one of the beams hit an obstacle then it is very likely that the other beam hit the obstacle too.

The key for successful updating is the threshold T which is composed of two elements, as described in lines 6-7. The first element is the scalar C , larger than unity, which is some function of the arrays C_{k-1} and C_{k-2} that contain information about the location of obstacles in the previous two scans, i.e., these arrays keep track of obstacle likelihood over time in the same way that the arrays T_{k-1} and T_{k-2} kept track of previous thresholdings in the GET-OBST-CANDIDATES routine. We will describe these arrays later, when we discuss the routine where they are updated. The second element of T is our approximation to the error of the estimate, i.e., a linear function that increases with range with parameters given by Eq. 1. Thus, a return adjacent to a candidate return is updated to be a candidate if the difference between their ranges lies within the expected error for that distance; if we have evidence that an obstacle has been sighted before at this location, then $C > 1$ relaxing the expected error.

V. Candidate pruning

The second three routines of CLASSIFY return a final classification R_f of each return based on pruning the array of candidates R_c . The objective of this pruning is to keep a low false alarm rate.

The first pruning exploits the time and space locality of the obstacles. The code of this routine is

```

REMOVE-NEW-OBST( $A, R_c$ )
1   for  $r \leftarrow 3$  to  $N_e - 2$ 
2      $i \leftarrow \lfloor A(r)/\alpha \rfloor + 1$ 
3     if  $\bigcap_{j=i-2}^{i+2} (C_{k-1}(j) = 0 \text{ and } C_{k-2}(j) = 0)$ 
4        $R_f(r) \leftarrow 0$ 
5     else
6        $R_f(r) \leftarrow R_c(r)$ 

```

7 **return** (R_f)

Lines 3-7 state that a candidate in $R_c(r)$ is a final obstacle in $R_f(r)$ only if an obstacle was located in the neighborhood in either of the previous two scans. An obstacle at a given location will be identified as a previously seen obstacle regardless of whether it moves slightly or is temporarily occluded. However, a previously seen obstacle will be identified as a new obstacle (and not added to R_f) if its position with respect to the lidar changes so fast that its new location is outside its previous neighborhood or if the occlusion lasts so long that the evidence of having seen it before has expired.

The second pruning routine, REMOVE-THIN-OBST, removes from the list of final obstacles R_f those returns that correspond to isolated hits, i.e., it keeps an obstacle only if there are at least O_s obstacle hits within a surrounding window of $\pm W_s/2$ returns, where $W_s \ll N_e$. The code of the routine is

```
REMOVE-THIN-OBST ( $R_f, W_s, O_s$ )
1   $d = \lfloor W_s/2 \rfloor$ 
2   $T[1 : N_e] \leftarrow R_f[1 : N_e]$ 
3  for  $r \leftarrow d + 1$  to  $N_e - d$ 
4      if  $T[r] > 0$ 
5           $cont \leftarrow 0$ 
6          for  $k \leftarrow -d$  to  $d$ 
7              if  $T[r + k] > 0$ 
8                   $cont \leftarrow cont + 1$ 
9          if  $cont < O_s$ 
10              $R_f[r] \leftarrow 0$ 
11 return ( $R_f$ )
```

The last pruning routine, CROP-RANGE, removes from R_f the returns that lie outside the range for which our linear approximation of the expected error holds, i.e., it zeros the values of $R_f(r)$ such that $R_f(r) < d_{min}$ or $R_f(r) > d_{max}$.

VI. Tracking the Scan Results

The routine UPDATE-AND-ARCHIVE updates the arrays C_{k-1} and C_{k-2} that keep track of the obstacles over time. The code of the routine is

```
UPDATE-AND-ARCHIVE ( $A, R_c$ )
1   $C_k(1 : N_s) \leftarrow 0$ 
2   $R_s(1 : N_s) \leftarrow 0$ 
3   $alarm \leftarrow 0$ 
4  for  $r \leftarrow 1$  to  $N_e$ 
5       $i \leftarrow \lfloor A(r)/\alpha \rfloor + 1$ 
6      if  $R_c(r) > 0$ 
7           $C_k(i) \leftarrow K_3$ 
8          if  $R_s(i) = 0$  or  $R_s(i) < R_c(r)$ 
9               $R_s(i) \leftarrow R_c(r)$ 
10              $alarm \leftarrow 1$ 
11  $C_{k-2} \leftarrow C_{k-1}$ 
12 for  $i \leftarrow 1$  to  $N_s$ 
13     if  $C_k(i) > C_{k-1}(i)$ 
14          $C_{k-1}(i) \leftarrow C_k(i) + C_{k-1}(i)$ 
15     else
16          $C_{k-1}(i) \leftarrow C_{k-1}(i) - \lceil (C_k(i) + C_{k-1}(i))/2 \rceil$ 
17         if  $C_{k-1} < 0$ 
```

18 $C_{k-1} \leftarrow 0$
19 **return** ($alarm, R_s$)

The first loop of the routine, in lines 4-10, accomplishes three things. First, it initializes the array C_k with a value K_3 in all the locations where a candidate was found. Second, in lines 8-9, it sets the summary array R_s to the smallest non-zero return, i.e., if there is an obstacle in the direction i , then $R_s(i)$ will contain the minimum distance between the robot and the obstacle. This value can be used to modify a robot trajectory without having to analyze individual returns. Finally, it raises the *alarm* flag if there is an obstacle in the scan.

Lines 11-18 update the arrays C_{k-1} and C_{k-2} that keep evidence of the presence of obstacles. The loop in lines 12-18 updates the vector C_{k-1} biasing it in a positive or negative direction depending on whether the evidence of the presence of an obstacle is stronger in this scan than it was in the previous scans, i.e., $C_k(i) > C_{k-1}(i)$. In our case, we have biased the evidence in the positive direction by adding the evidences of C_k and C_{k-1} . This is a strong bias that makes the algorithm aware of the presence of an obstacle very fast. In contrast, we biased the evidence in the negative direction by subtracting from C_{k-1} the average of the evidence of C_k and C_{k-1} . This is a weak bias that makes the algorithm forget the presence of an obstacle very slowly. Thus, obstacles that have been seen in the past, even after many scans, are rapidly identified when they reappear. Lines 17-18 make sure that the evidence is never negative.

VII. Experimental results

The algorithm was designed using the data set collected by Jose Macedo and Mike McHenry to support a related work [5]. The set consists of eight sequences with a length that varies between 15 and 90 seconds. The platform used to gather this set was the Urbie robot, a small track-based platform shown in Fig. 3, using a custom lidar based on the Accuity AccuRange 4000 [7], [8]. This lidar operates at a wavelength of 1064 nm, in the near-visible infrared, and has a sampling interval of $\theta = 10$ mrad (i.e., 0.6 degrees). Each scan has $N_e = 512$ returns and thus, it covers an angle of $\Omega = 307^\circ$. The lidar is capable of estimating range up to distances of 15 m (i.e., 50 ft).

The algorithms have a number of parameters that, in practice, are easy to adjust and once set, work well in a wide variety of scenarios. Given the size and speed of Urbie, we set the range of interest to lie within $d_{min} = 0.3$ m and $d_{max} = 2$ m. We measured the error of the estimates of two known targets within this range and estimated the coefficients in Eq. 1 to be $a = 0.015$ and $b = 12.3$. This means that we expect the error to rise at a rate of 1 mm for every 70 mm of range. The free parameters in GET-OBS-CANDIDATES were set to $K_1 = 0.025$ and $\delta = 0.03$. The free parameter in RELAX-OBST was set to $K_2 = 0.05$. The parameters of REMOVE-THIN-OBST were set to $W_s = 21$ and $O_s = 8$ so we eliminate returns that do not hit the obstacle at least 8 times in a span of 15° . Finally, the free parameter in UPDATE-AND-ARCHIVE was set to $K_3 = 25$.

Once the parameters of the algorithm were set, they were used in all the scenarios of the data set. The results for this set were 4 false alarms (i.e., a return from grass misclassified as a return from an obstacle) in 1313 images which covered scenarios where the robot was in the clear and in grass, both sparse and dense, both while standing still and moving, and with and without rocks, both in hidden by grass and unoccluded. Figure 3 shows the robot approaching two rocks partially hidden by sparse grass. The corresponding scan produced by the laser for an area of $2 \times 2 m^2$ is shown in Fig. 4. The laser is located at the origin of the graph. In this top view, the front of the robot corresponds to the upwards direction. The solid pie slices originating from the laser are robot self-occlusions caused by the on-board camera, the antennas and other structures. The long lines originating from the laser indicate returns that were identified as NOTFOLIAGE. All other returns (i.e., the dots of the graphs) were identified as FOLIAGE.

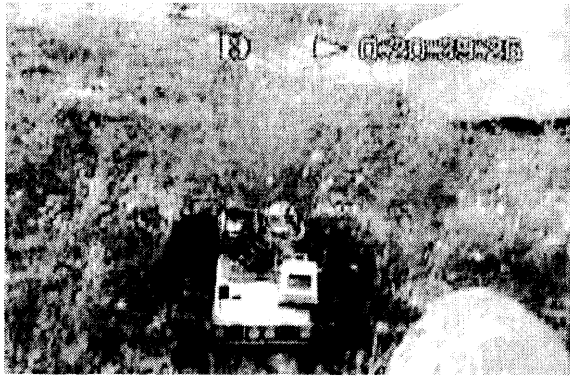


Fig. 3. Robot facing two rocks at 45° and 315°

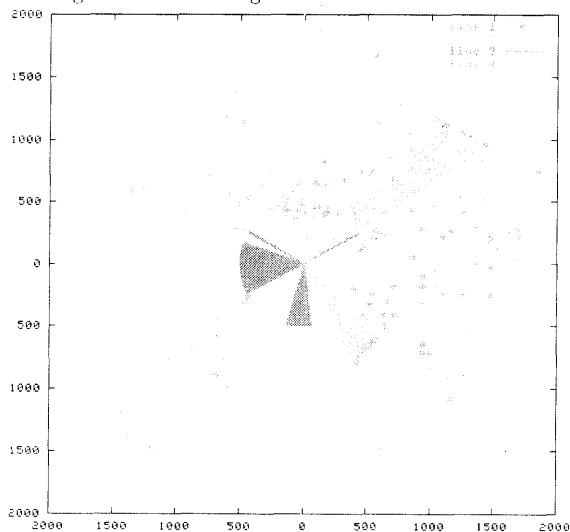


Fig. 4. Return classification of scene in Fig. 3

Once the prototyping of the algorithm was finished, it was implemented in C++ and ported to Urbie, which runs under VxWorks. At the time of the porting, Urbie had a Sick lidar with a sampling interval of $\theta = 8.7$ mrad (i.e., 0.5 degrees) that spanned an

angle of $\Omega = \pi$ in each scan and had an expected range error 70 mm at 4 m. The porting of the final code to VxWorks was done in a few hours and the algorithm was easily integrated to the real-time system of Urbie and tested successfully in dense grass with hidden obstacles.

VIII. Conclusion

We have presented a real-time algorithm to identify foliage present in a natural scene. Once the foliage is identified, it can be removed from the obstacle list and the result can be taken into account to reevaluate the traversability of a path. The algorithm uses the locality of obstacles in time and space to first identify a list of returns that are likely to be obstacles and then to prune this list to eliminate false positives. The algorithm was successfully ported to the Urbie robot and performed correctly during test on dense foliage scenarios. Our future plans are to extend this algorithm to handle range images and to include the intensity of the return in making the classification decision.

Acknowledgments

Thanks to Jose Macedo and Roberto Manduchi for useful discussions and to Mike McHenry for porting the final code to VxWorks. The research in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The collection of data and the prototyping of the algorithm were supported by DARPA, Mobile Autonomous Robot Software Program, under contract NAS7-1407, task order 15276. The real-time implementation and porting to the Urbie robot were supported by DARPA, Tactical Mobile Robotics Program, under contract NAS-1407, task order 15089.

References

- [1] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the carnegie-mellon navlab," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 10, pp. 362–373, May 1988.
- [2] M. Adams, "Lidar design, use, and calibration concepts for correct environmental detection," *IEEE Trans. Robotics Automat.*, vol. 16, pp. 753–761, Dec. 2000.
- [3] M. Hebert and E. Krotkov, "3-d measurements from imaging laser radars: how good are they?," *Intl. Journal Image and Vision Computing*, vol. 10, pp. 170–178, Apr. 1992.
- [4] I. S. Kweon, R. Hoffman, and E. Krotkov, "Experimental characterization of the perceptron laser rangefinder," Tech. Rep. CMU-RI-TR-91-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Jan. 1991.
- [5] J. Macedo, R. Manduchi, and L. Matthies, "Ladar-based discrimination of grass from obstacles for autonomous navigation," in *Proc. Intl. Symposium on Experimental Robotics*, pp. 111–120, 2000.
- [6] J. Huang, A. B. Lee, and D. Mumford, "Statistics of range images," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 324–331, 2000.
- [7] L. Matthies, Y. Xiong, R. Hogg, D. Zhu, A. Rankin, B. Kennedy, M. Hebert, R. MacLachlan, C. Won, T. Frost, G. Sukhatme, M. McHenry, and S. Goldberg, "A portable autonomous urban reconnaissance robot," in *Proc. 6th Intl. Conf. Intelligent Autonomous Systems*, 2000.
- [8] C. Bergh, B. Kennedy, L. Matthies, and A. Johnson, "A compact low power two-axis scanning laser rangefinder for mobile robots," in *Proc. 7th Mechatronics Forum International Conf.*, 2000.